	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorio de Computacion Salas A y B

Profesor(a): Rene Adrian Davila Perz

Asignatura: Programacion orientada a objetos

Grupo: 01

No de practica(s): 03

Integrante(s): 322001549, 322006829, 322125533, 425131152, 322103351

No de lista o brigada: 07

Semestre: 2026-1

Fecha de entrega: Miercoles 10 de septiembre de 2025

Observaciones:

Calificacion:

Índice

1. Introducción

1.1. Planteamiento del Problema

La practica consistió en implementar en Java un programa que simule un algoritmo de función digestiva, osea que genere claves apartir de argumentos dados por el usuario al momento de ejecutar. Esto permite aplicar las librerias de ArrayList y HashMap, mejorar el uso del lenguaje y expandir la aplicación de librerías en la programación.

1.2. Motivación

Practicar con estos algoritmos fortalece el entendimiento de las funciones digestivas, generación de claves y usos del Hash. Además nos permite saber como usar el formato de impresión y ejecución de claves Hash aunque son teóricas.

1.3. Objetivos

- Implementar en Java el algoritmo de generaHash.
- Simular un MD5.
- Diseñar un código que use el generaHash para generar las claves aleatorias de los argumentos.
- Documentar la solución teórica, la implementación y los resultados obtenidos.

2. Marco teórico

2.1. ArrayList y diferencias con Array

El ArrayList de Java tiene un tamaño dinámico, lo que significa que los elementos pueden ser fácilmente añadidos o eliminados. Además, la clase ArrayList pertenece al Java Collections Framework y, a diferencia de los arrays, no está disponible de forma nativa. Debe importarse de la biblioteca java.util.[1]

El ArrayList es una opción apropiada cuando la longitud de la lista de Java puede variar. Algunos ejemplos son el almacenamiento de objetos, la búsqueda u ordenación de datos y la creación de listas o colas.[1]

El tamaño del tipo de datos array no puede modificarse. Por lo tanto, el número de objetos que debe contener el array debe conocerse de antemano. Por ello, los arrays son adecuados para gestionar un conjunto predefinido de tipos de datos primitivos como int, float, chat o boolean.[1]

Una desventaja de ArrayList es el mayor tiempo de acceso. Mientras que con los arrays existe una zona de memoria reservada fija, con ArrayList esta no es contigua. Por lo tanto, es importante tener en cuenta las ventajas y desventajas respectivas y seleccionar la estructura de datos adecuada para cada uso. Antes de crear un ArrayList, la clase correspondiente debe ser importada de la librería java.util:[1]

```
import java.util.ArrayList;
```

La sintaxis general es:

```
ArrayList<Type> arrayList= new ArrayList<>();
```

“Type” representa el tipo de datos respectivo de Java ArrayList. A continuación, creamos listas de tipo String e Integer.[1]

```
ArrayList<String> arrayList= new ArrayList<>();
```

```
ArrayList<Integer> arrayList= new ArrayList<>();
```

ArrayList utiliza la clase wrapper correspondiente de los tipos de datos primitivos para que sean tratados como objetos. Por lo tanto, necesitamos especificar Integer en lugar de int.[1]

Una vez creada la ArrayList “colors” de tipo String, añadimos varios elementos con el método .add().[1]

```
import java.util.ArrayList;
class Main {
    public static void main(String[] args){
        ArrayList<String> colors = new ArrayList<>();
        colors.add("blue");
        colors.add("red");
        colors.add("green");
        System.out.println("ArrayList:" + colors);
    }
}
```

Esto da este resultado:

```
ArrayList: [blue, red, green]
```

Para eliminar objetos de un ArrayList Java, utilizamos el método .remove() con la especificación del índice del elemento:

```
import java.util.ArrayList;
class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("blue");
        colors.add("red");
        colors.add("green");
        System.out.println("ArrayList:" + colors);
        String color = colors.remove(1);
        System.out.println("ArrayList:" + colors);
        System.out.println("Removed Element:" + color);
    }
}
```

Resultado:

```
ArrayList: [blue, green]
Removed Element: red
```

Con la función .get() accedemos a un elemento en una posición concreta:

```
import java.util.ArrayList;
class Main {
    public static void main(String[] args) {
        ArrayList<String> clothes = new ArrayList<>();
        clothes.add("jacket");
        clothes.add("shirt");
        clothes.add("pullover");
        System.out.println("ArrayList: " + clothes);
        String str = clothes.get(2);
        System.out.print("Element at index 2: " + str);
    }
}
```

Resultado:

```
ArrayList: [jacket, shirt, pullover]
Element at index 2: pullover
```

El número de elementos de un ArrayList se puede calcular fácilmente con el método .size():

```
import java.util.ArrayList;
class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("blue");
        colors.add("red");
        colors.add("green");
        System.out.println(colors.size());
    }
}
```

resultado:

```
3
```

2.2. HashMap

Existen distintos métodos para almacenar y recuperar datos. Dependiendo del uso que se le quiera dar, usar uno u otro tipo de almacenamiento puede resultar más conveniente. En muchos casos, optar por la clase HashMap de Java puede ser la mejor solución. A diferencia de otros métodos, esta clase almacena los datos como pares clave-valor, es decir, a cada clave se le asigna exactamente un valor. Para recuperar el valor, basta con usar la clave correspondiente. Las claves y los valores pueden ser tipos de datos muy diferentes, como cadenas, números u otros objetos.[2]

La clase HashMap de Java tiene, por tanto, varias ventajas. En primer lugar, permite realizar una búsqueda rápida dentro del marco del lenguaje de programación. Al mismo tiempo, el enfoque clave-valor evita que a una misma clave se le asignen múltiples valores y de esta forma, se eluden las duplicaciones. Solo los objetos se pueden incluir varias veces con claves diferentes. En lo que respecta al rendimiento, este tipo de almacenamiento y búsqueda también resulta ventajoso si se compara, por ejemplo, con las listas rígidas, que son mucho menos flexibles. Por ello, las bases de

datos claves-valor siguen este principio. A continuación, descubre cómo crear HashMaps en Java y utilizarlos para tus propósitos.[2]

Para poder mostrar mejor el funcionamiento de Hashmap en Java, se presenta un caso práctico: una empresa quiere guardar una lista de clientes que pueda consultar en cualquier momento. Se necesita, por un lado, el nombre del cliente y, por otro, un número de cliente. Aunque el número de cliente (se corresponde con la clave) siempre es único, en teoría podrían existir varios clientes con el mismo nombre (se almacenaría como valor). El número se almacena como interger y los nombres como strings. En Java, HashMap se vería:[2]

```
import java.util.HashMap;
public clase Main {
public static void main(String[] args) {
    HashMap<Integer, String> lista de clientes = new HashMap<>();
    }
}
```

Aunque ya tenemos el HashMap de Java, se encuentra vacío. Para añadir nuevos pares clave-valor, se usa el método put(). A continuación el ejemplo:

```
public clase Main {
public static void main(String[] args) {
        HashMap<Integer, String> lista de clientes = new HashMap<>();
        lista de clientes.put (1077, "Manolo Rodriguez");
        lista de clientes.put (15312, "Pepe Garcia");
        lista de clientes.put (73329, "Maria Perez");
        System.out.println(kundenliste);
        }
}
```

Con el comando System.out.println se imprime la lista de clientes:

```
{1077=Manolo Rodriguez, 15312=Pepe Garcia, 73329=Maria Perez}
```

Aunque ya se dispone de una lista de clientes a la que teóricamente se le pueden añadir muchas más entradas, también puedes querer acceder a un cliente individual. Para ello, aplica el método get() a la clave correspondiente. Se vería:[2]

```
public clase Main {
public static void main(String[] args) {
        HashMap<Integer, String> lista de clientes = new HashMap<>();
        lista de clientes.put (1077, " Manolo Rodriguez");
        lista de clientes.put (15312, "Pepe Garcia");
        lista de clientes.put (73329, "Maria Perez");
        System.out.println(lista de clientes.get(1077));
        }
}
```

Como resultado, solo debería aparecer “Manolo Rodríguez”. Si deseas eliminar una entrada específica, recurre al método remove(). Para ponerlo en práctica, sigue el ejemplo:[2]

```
public clase Main {
public static void main(String[] args) {
        HashMap<Integer, String> lista de clientes = new HashMap<>();
        lista de clientes.put (1077, "Manolo Rodriguez");
        lista de clientes.put (15312, "Pepe Garcia");
        lista de clientes.put (73329, "Maria Perez");
        lista de clientes.remove(1077);
        System.out.println(lista de clientes);
        }
}
```

En la consola, se mostraría:

```
{15312=Pepe Garcia, 73329=Maria Perez}
```

También es posible acceder a una lista que solo contenga claves o valores. Para ello, hay que recurrir a un bucle for-each. Utiliza el método keySet() para las claves y el método values() para los valores. Con este último, el código se mostraría como sigue:

```
public class Main {
public static void main(String[] args) {
        HashMap<Integer, String> lista de clientes = new HashMap<>();
        lista de clientes.put (1077, "Manolo Rodriguez");
        lista de clientes.put (15312, "Pepe Garcia");
        lista de clientes.put (73329, "Maria Perez");
        for (String i : lista de clientes.values()) {
                System.out.println(i);
                }
        }
}
```

Se mostrarían los siguientes resultados:

```
Manolo Rodriguez
Pepe Garcia
Maria Perez
```

No solo es posible acceder a una entrada concreta, sino que también se puede consultar si el HashMap de Java contiene algún valor o clave concretos. Para ello, se utilizan los métodos containsKey() (para claves) y containsValue() (para valores). Si el elemento está incluido, se obtiene “true”. Si el elemento no está incluido, el resultado es “false”. En la práctica, los métodos se implementan como se muestra a continuación:[2]

```
public class Main {
public static void main(String[] args) {
        HashMap<Integer, String> lista de clientes = new HashMap<>();
        lista de clientes.put (1077, "-Manolo-Rodriguez");
        lista de clientes.put (15312, "Pepe-Garcia");
        lista de clientes.put (73329, "Maria-Perez");
        System.out.println(lista de clientes.containsKey(15312));
        System.out.println(lista de clientes.containsValue("Ana-Martinez");
        }
}
```

Aunque en la lista sí aparece la clave "15312", el valor "Ana Martínez" no aparece, por lo que la consola mostrará como resultado:[2]

```
true
false
```

3. Desarrollo

3.1. Descripción de la Implementación

En este practica se implemento un pequeño simulador de funciones hash en Java. Ademas de implementar por primera vez en el curso las librerías de ArrayList y HashMap, donde ArrayList es una estructura de datos dinámica que permite almacenar listas de elementos que pueden crecer o reducirse de tamaño de manera automática que en este caso se utilizó para guardar las palabras que se reciben como argumentos desde la terminal., por otro lado, HashMap es una estructura de datos que almacena pares clave–valor, permitiendo acceder rápidamente a los valores a través de una clave única el cual se utilizó para relacionar el hash generado (clave) con la palabra original (valor).

La idea principal consiste en que el programa recibe como argumentos una serie de palabras o cadenas, y para cada una de ellas se genera una clave hash única simulada.

El hash se genera de la siguiente forma:

1. Se calcula una semilla a partir de la suma de los valores ASCII de todos los caracteres de la palabra.
2. Con la semilla se inicializa un objeto de tipo random lo que asegura que no se obtenga la misma clave.
3. Se generan 32 números aleatorios en el rango de 0 a 15.
4. Cada número se convierte a su representación hexadecimal.
5. Finalmente, los 32 valores hexadecimales se concatenan para formar la clave hash simulada.

Posteriormente, cada par (hash, palabra original) se almacena en la estructura HashMap, donde la clave es el hash y el valor es la palabra o palabras originales. Finalmente, se muestran en pantalla las claves generadas de cada palabra o cadena.

3.2. Estructura del programa

La lógica del simulador puede representarse de la siguiente forma:

Pseudocodigo

```
Inicio
  Leer argumentos desde la terminal
  Guardar argumentos en un ArrayList
  Para cada palabra:
       Generar hash con generaHash()
       Insertar en HashMap (clave=hash, valor=palabra)
  Mostrar claves generadas en la terminal
Fin
```

3.3. Pruebas Realizadas

Para la practica realizamos diferentes pruebas, debido a que el código al ejecutarlo con los mismos argumentos no generaba la misma clave hash, hasta que cerramos, volvimos ha abrir la terminal, compilamos y ejecutamos de nuevo, ahí nos dio una clave de salida diferente para los argumentos anteriormente utilizados.

Para ejecutar se compila y al momento de llamar el código se le pone los argumentos de los que se desea generar una clave:

java Practica3 Hola

Clave:c9d3ef43eaa48e1e0a8fde0a176039ac ——— Valor: Hola

```
PS C:\Users\52551\Programacion\PracticasLab-3\Practica3> javac .\Practica3.java
PS C:\Users\52551\Programacion\PracticasLab-3\Practica3> java Practica3
PS C:\Users\52551\Programacion\PracticasLab-3\Practica3> java Practica3 Hola Mundo
Clave:a4ef0e40c9220e0eabc2b7dc7ef06ddf    Valor: Mundo
Clave:c9d3ef43eaa48e1e0a8fde0a176039ac    Valor: Hola
PS C:\Users\52551\Programacion\PracticasLab-3\Practica3> java Practica3 Simulador de hash en Java
Clave:b9afc032e93db583ae5fdd9564948508    Valor: Simulador
Clave:b08b762398d585ab514f6a643ee171ae    Valor: en
Clave:c33d2358c62106a2306eb20fdbda7877    Valor: hash
Clave:b0e845853a5c98792321841d03783da7    Valor: de
Clave:cfc6899fb19d1a57a48115be377dd6da    Valor: Java
PS C:\Users\52551\Programacion\PracticasLab-3\Practica3>
```

Como se menciono anteriormente primero compilamos el código y llamamos a la función con los argumentos después del java Practica3, como hola mundo.

4. Resultados

```
Run | Debug
public static void main(String[] args) {
    ArrayList<String> palabra=new ArrayList<String>();
    for(int i=0; i<args.length;i++){
        palabra.add(args[i]);
    }
    HashMap<String,String>Datos=new HashMap<>();
    for(int j =0;j<palabra.size();j++){
        Datos.put(generaHash(palabra.get(j)), palabra.get(j));
    }
    for (var entry: Datos.entrySet() ){
        System.out.println("Clave:" + entry.getKey() + "    Valor: " + entry.getValue());
    }

}
```

En la función principal (main) del código al cual funciona mediante argumentos String donde definimos palabra como un ArrayList de cadenas donde el primer ciclo for recorre todos los argumentos que entraron en la consola(terminal) y se añade al ArrayList palabra. Después se crea un HashMap llamado datos que va a contener el hash generado y la palabra el cual se utiliza en el segundo ciclo for que va generando el hash de cada palabra. Finalmente, el ultimo ciclo for se utiliza para recorrer e imprimir todos los elementos del HashMap datos y les da su formato de impresión.

```
public static String generaHash(String texto) {
    // Crear la semilla a partir de la suma de los caracteres
    int semilla = 0;
    for (char c : texto.toCharArray()) {
        semilla += c;
    }
            Random random - Practica3.generaHash(String)
    Random random = new Random(semilla);

    // Generar 32 caracteres hexadecimales (simulando MD5)
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 32; i++) {
        int valor = random.nextInt(bound:16); // 0 - F
        sb.append(Integer.toHexString(valor));
    }

    return sb.toString();
}
}
```

En la función generaHash se crea una variable semilla que inicializa en 0 la cual se ira modificando en el primer ciclo for a partir de la suma de los caracteres. se crea la función random que estará en conjunto con la semilla y con el StringBuilder definido en segundo ciclo for se generaran los 32 caracteres hexadecimales de la palabra que se le pase. Esta función se ira llamando en la función Main tantas veces hasta que haya generado el Hash de cada palabra que entraron en la consola(terminal).

5. Conclusiones

La práctica permitió comprobar que los conceptos relacionados con la generación de claves hash pueden trasladarse a un entorno de programación en Java mediante el uso de estructuras de datos y funciones personalizadas. El simulador implementado generó claves a partir de cadenas de texto, utilizando como base la suma de los caracteres y un generador pseudoaleatorio inicializado con esta semilla. De esta manera, se emuló el funcionamiento de un algoritmo de hash, mostrando cómo una entrada puede transformarse en una salida única de longitud fija.

La implementación de las clases ArrayList y HashMap, utilizadas por primera vez en este proyecto, resultó fundamental para organizar los datos. Mientras que el ArrayList permitió almacenar dinámicamente las palabras ingresadas como argumentos, el HashMap facilitó la asociación entre cada palabra y su clave hash simulada, funcionando como una tabla de correspondencias. Esto no solo permitió reforzar la comprensión de estas estructuras de la biblioteca estándar de Java, sino también demostrar su utilidad práctica en la solución de problemas.

Las pruebas realizadas confirmaron que la misma palabra siempre genera la misma clave hash y que, en caso de palabras repetidas, el HashMap evita la duplicación de entradas. Asimismo, se observó que diferentes cadenas producen claves distintas, cumpliendo con el objetivo de la práctica de simular un mecanismo de dispersión de datos.

En este sentido, se alcanzaron los objetivos planteados: implementar un programa en Java que genere claves hash simuladas, explorar por primera vez el uso de ArrayList y HashMap, y verificar su funcionamiento a través de pruebas en consola. El ejercicio evidencia cómo los conceptos teóricos de estructuras de datos y funciones hash pueden materializarse en código, fortaleciendo la lógica algorítmica y la comprensión de la programación en Java.

Referencias

[1] I. editorial team. (2023, Mar.) How to use java arraylist. Accedido: 8 de septiembre de 2025. [Online]. Available: https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/java-arraylist/

[2] E. editorial de IONOS. (2024, diciembre) Java hashmap: almacenar datos en pares clave-valor. Accedido: 8 de septiembre de 2025. [Online]. Available: https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/java-hashmap/